Zagadnienia Sztucznej Inteligencji – laboratorium

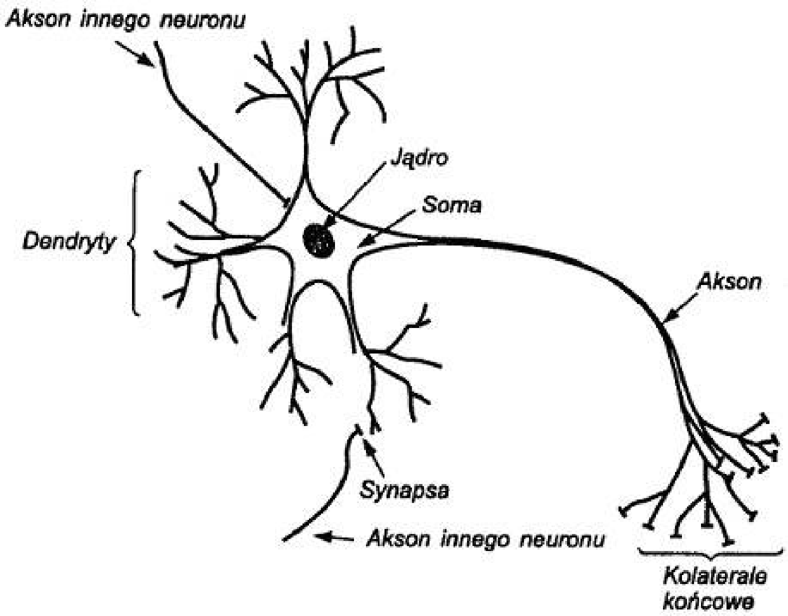
Sieci Neuronowe 1

Michał Bereta

Wprowadzenie

Sztuczne sieci neuronowe można postrzegać jako modele matematyczne, które swoje wzorce wywodzą z biologii i obserwacji ludzkich komórek nerwowych. Pierwszy model neuronu został zaproponowany w 1943 roku przez McCullocha i Pittsa. Był to prosty neurony, który mógł modelować funkcje logiczne takie jak OR lub AND. W 1958 roku Rosenblatt zaproponował model perceptronu, a w roku 1960 powstał model ADALINE autorstwa Widrowa i Hoffa.

Model biologicznej komórki nerwowej pokazano na rysunku 1.



Rysunek 1: Model komórki nerwowej .

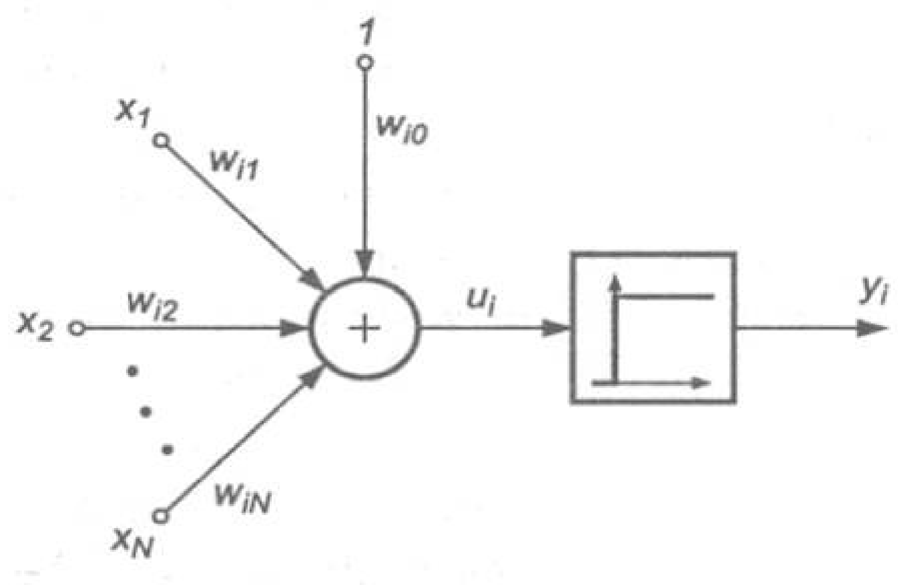
Do każdego neuronu dochodzą pewne sygnały (bodźce). Są one doprowadzane za pomocą synaps. Bodźce pochodzące od synaps wywoływane są przez specjalne substancje chemiczne, zwane neuromediatorami. Zmiana potencjału elektrycznego komórki zależna jest od ilości neuromediatora w synapsie. Sygnał wyjściowy jest wyprowadzany z komórki za pomocą aksonu. Sygnały wyjściowe neuronów mogą być sygnałami wejściowymi innych neuronów. Sygnałami pobudzającymi neurony mogą być także sygnały pochodzące z receptorów nerwowych.

Podstawowe pojęcia i definicje

Na podstawie biologicznego modelu z rysunku 1, McCulloch i Pitts, zaproponowali model matematyczny sztucznego neuronu (rysunek 2).Sygnały dochodzące do neuronu (x_1, x_2, ...x_N) mnożone są przez tzw. wagi w_{ij} (***i*** - numer neuronu; ***j*** - numer wagi). Parametr ***k*** oznacza chwile czasową. Dodatkowy sygnał jednostkowy nazwano biasem. Na podstawie wag i dochodzących sygnałów obliczany jest stan wewnętrzny neuronu u_i

$$u_i(k) = \sum_{j=1}^{N} w_{ij}(k) x_j(k) + w_{i0}(k)$$

Wzór 1: Aktywacja perceptronu (sumacja ważona).



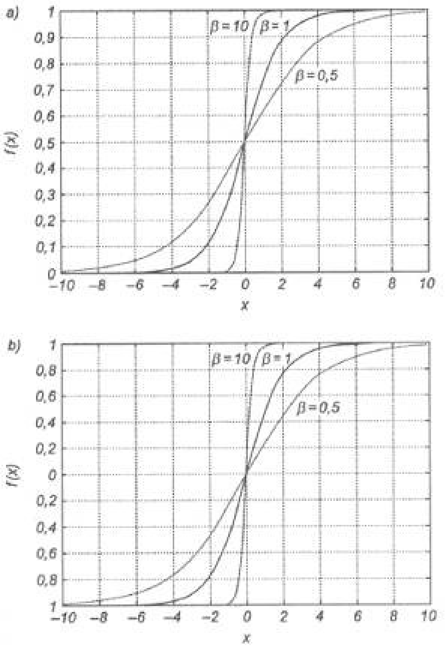
Rysunek 2: Model sztucznego neuronu McCullocha-Pittsa.

Natomiast odpowiedź neuronu y_i zależy od tego czy stan wewnętrzny neuronu (jego pobudzenie) przekracza pewien poziom. Odpowiada za to funkcja aktywacji $f(u_i)$**,** która w modelu McCullocha-Pittsa jest funkcją progową.

$$f(u_i) = \begin{cases} 1 & u_i > 0 \\ 0 & u_i \leq 0 \end{cases}$$

Wzór 2: Unipolarna funkcja aktywacji perceptonu.

Najczęściej stosowanymi funkcjami aktywacji neuronu są (oprócz funkcji liniowej) funkcja sigmoidalna oraz tangens hiperboliczny (rysunek 3).



$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

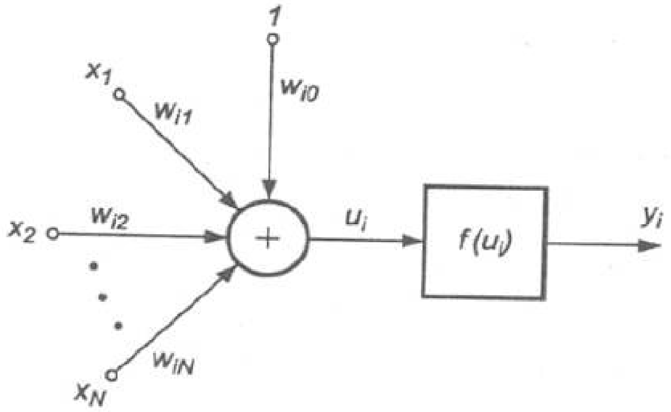
$$f(x) = \tanh(\beta x)$$

Rysunek 3: Funkcje aktywacji: sigmoidalna i tangens hiperboliczny.

Parametr **β** pozwala wpływać na kształt funkcji. Popularność tych funkcji spowodowana jest łatwością obliczania ich pochodnych, co jest konieczne przy użyciu algorytmów uczenia opartych na podejściu gradientowym. Wartości pochodnych dla powyższych funkcji łatwo obliczyć znając jedynie wartości samych funkcji. Stan wewnętrzny (pobudzenie neuronu) tak jak poprzednio oblicza się sumując z wagami sygnały wejściowe neuronu:

$$u_i(k) = \sum_{j=1}^{N} w_{ij}(k) x_j(k) + w_{i0}(k)$$

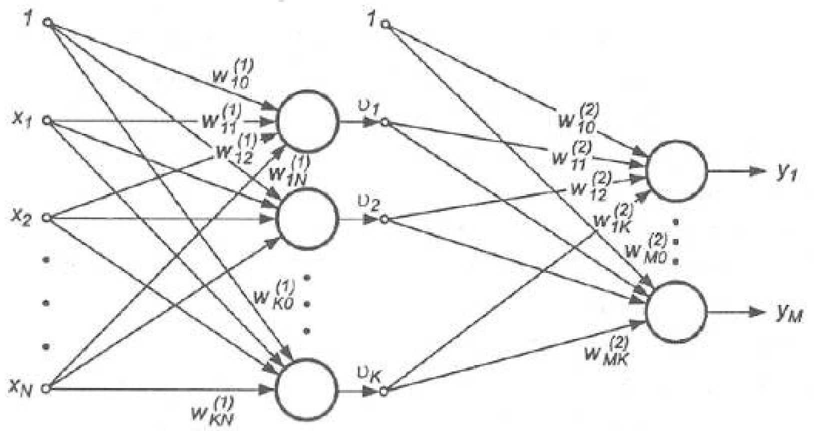
Sygnał wyjściowy neuronu jest wyliczany za pomocą funkcji aktywacji sigmoidalnej, ***tanh*** lub innej, która jako argument przyjmuje stopień pobudzenia neuronu (rysunek 4).



Rysunek 4: Model sztucznego neuronu z nieliniową funkcją aktywacji.

Sztuczna sieć neuronowa

Sztuczną sieć neuronowa uzyskuje się łącząc ze sobą warstwy neuronów. Na rysunku 5 pokazano model sieci wielowarstwowej.

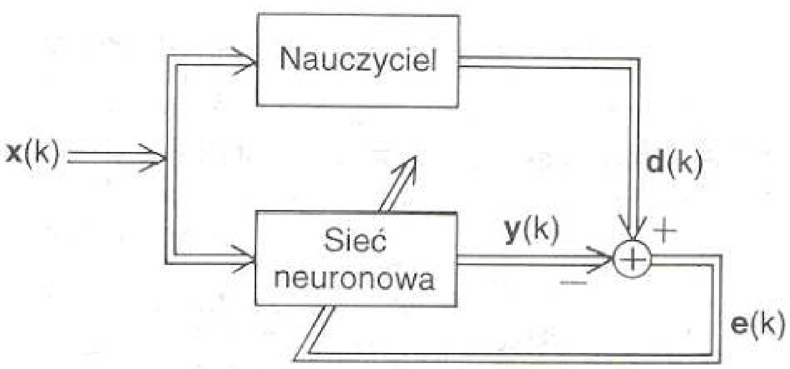


Rysunek 5: Model sieci z dwiema warstwami.

Na rysunku 5 przedstawiono sieć neuronową dwuwarstwową.. Widać, że sygnały wyjściowe warstwy pierwszej są jednocześnie sygnałami wejściowymi neuronów warstwy drugiej, która jest jednocześnie warstwą wyjściową sieci, tzn. odpowiedzi neuronów tej warstwy określają odpowiedź całej sieci. Dodawanie nowych warstw powoduje, że sieć ma większą moc obliczeniową. W wyniku wykorzystania wielu warstw, sieć potrafi poradzić sobie z problemami, które mają charakter nieliniowy, czego nie jest w stanie dokonać sieć jednowarstwowa.

Uczenie sieci

Uczenie pod nadzorem. Uczenie sieci jest to proces doboru odpowiedniego zestawu wag danej sieci, dla danego konkretnego zadani. W zasadzie istnieją dwie metody uczenia sieci: bez nauczyciela i z nauczycielem (pod nadzorem)- rysunek 6.



Rysunek 6: Schemat uczenia pod nadzorem.

W metodzie uczenia pod nadzorem na wejścia sieci podaje się sygnały ***x***. Sieć odpowiada sygnałami ***y***, które są wyjściami poszczególnych neuronów. Nauczyciel, wiedząc jakie sygnały są pożądane (sygnały ***d***) na wyjściu sieci dla każdego sygnału ***x***, porównuje je z aktualną odpowiedzią sieci, po czym modyfikuje wagi neuronów z uwzględnieniem znanego błędu ***e*** sieci. Dobór odpowiednich wag odpowiada procesowi minimalizacji funkcji błędu, którą można zapisać jako:

$$E = \sum_{i=1}^{N} (y_i(k) - d_i(k))^2$$

gdzie ***N*** to liczba neuronów w warstwie wyjściowej sieci (sumowane są błędy wszystkich neuronów wyjściowych). Sumaryczny błąd uwzględniający wszystkie prezentowane sieci wzorce (lub ich część) można zapisać jako:

$$E = \sum_{i=1}^{N} \sum_{k=1}^{p} (y_i(k) - d_i(k))^2$$

Błąd ten jest równy sumie błędów wszystkich neuronów (sumowanie po ***i***) ze względu na wszystkie prezentowane im dane uczące (sumowanie po ***k***). Modyfikacja wag odbywa się według wzoru:

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$

gdzie ***i*** oznacza numer neuronu, ***j*** oznacza kolejną wagę tego neuronu, a ***k*** oznacza krok czasowy. Konkretna wartość Δw_{ij} zależy od przyjętego algorytmu uczenia. Do uczenia sieci często używa się **metod gradientowych** – obliczając gradient funkcji błędu a następnie modyfikując wartości wag w kierunku największego spadku funkcji błędu. Do doboru wartości wag wykorzystuje się również algorytmy genetyczne.

Uczenie bez nadzoru. Niektóre rodzaje sieci neuronowych, jak na przykład sieci Hebba czy Kohonena, uczone są bez wykorzystania zbioru wzorcowych odpowiedzi. Sieć uczona jest za pomocą jedynie danych wejściowych. Sieci tego typu wykorzystywane są min. w zadaniach klasteryzacji (grupowania). Klasteryzacja to dzielenie danego zbioru danych na podzbiory, których elementy maja pewne cechy wspólne.

Zastosowanie sieci neuronowych

Sztuczne sieci neuronowe stosuje się do takich zagadnień jak:
- klasyfikacja danych
- aproksymacja funkcji
- predykcja szeregów czasowych
- rozpoznawanie wzorców
- kompresja danych

Podstawowe modele sieci neuronowych

Perceptron prosty

Model McCullocha-Pittsa przedstawiony na rysunku 2 jest punktem wyjścia do konstrukcji najprostszej sztucznej sieci neuronowej o nazwie ***perceptron*** (zwanej również ***perceptronem prostym***). Zadaniem perceptronu prostego jest klasyfikacja podanego na wejście wektora ***x*** do jednej z dwoch klas L_1 lub L_2. Jeśli sygnał wyjściowy neuronu przyjmuje wartość 1 to wektor ***x*** jest zaklasyfikowany do klasy L_1, jeśli przyjmuje wartość 0 – do klasy L_2. Zatem perceptron dzieli ***N***-wymiarową przestrzeń wektorów wejściowych na dwie półprzestrzenie rozdzielone ***N-1*** wymiarową hiperpłaszczyzną. Hiperpłaszczyzna ta zwana jest powierzchnią (granicą) decyzyjną. Jeśli ***N=2*** to jest to linia prosta. Aby zbiór danych wejsciowych mógł by oddzielony przez hiperpłaszczyznę, musi być liniowo separowalny. Nieznane wartości wag dobierane są w procesie uczenia perceptronu.

Reguła uczenia perceptronu

Jeśli na wejście neuronu podawane są wektory $\mathbf{x=[1,x_1, x_2, ...]}$, sieć odpowiada sygnałem ***y***, a żądana odpowiedź jest równa ***d***, to wektor wag $\mathbf{w=[w_0,w_1,w_2,...]}$ jest modyfikowany zgodnie z poniższymi zasadami:

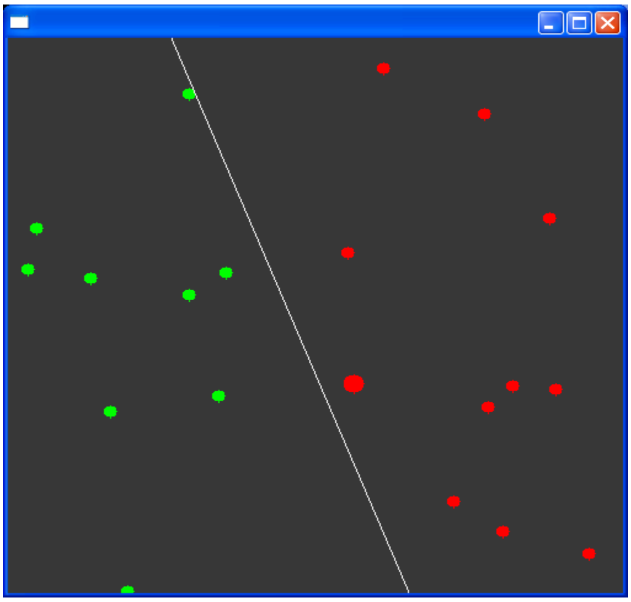
- jeśli $y=d$ to wagi pozostają niezmienione (wektor x został prawidłowo zaklasyfikowany)
- jeśli $y=0$ a $d=1$ to $w_j(k+1) = w_j(k) + \eta * x_j$
- jeśli $y=1$ a $d=0$ to $w_j(k+1) = w_j(k) - \eta * x_j$

gdzie η jest krokiem korekcji. Procedurę tę powtarza się dla wszystkich danych uczących **(x,d)** wielokrotnie. Kroki te można zapisać łącznie jako

$$w_j(k+1) = w_j(k) + \eta*(d-y)*x_j$$

Zadanie do wykonania: Uczenie perceptronu

Należy pobrać ze strony demonstracyjny program ***PerceptronDemo***. Zaobserwować jak dwie klasy liniowo separowane są oddzielone przez powierzchnię decyzyjną (linię prostą w tym przypadku). Uczony perceptron ma bipolarną (odpowiada wartościami 1 lub -1) funkcje aktywacji. W trakcie wykonywania programu, dane uczące mogą być modyfikowane poprzez ich przeciąganie z przyciśniętym lewym przyciskiem myszki. Po kliknięciu i przytrzymaniu prawego przycisku myszki, wagi perceptronu zostaną ustawione na losowe wartości, tak aby proces uczenia mógł zacząć sie ponownie. Okno programu predstawia wycinek płaszczyzny 2d o wymiarach [0 ; 1] x [0 ; 1], punkt (0,0) znajduje się w lewym dolnym rogu.



Rysunek 7: Program PerceptronDemo

Po uruchomieniu programu na oknie konsoli są widoczne wartości parametrów, z jakimi program został uruchomiony. Są to: krok korekcji ***η (eta)*** oraz flaga wskazująca czy w trakcie nauki używany jest bias. Parametry te moża ustawić w trakcie uruchamiania programu z lini poleceń, np:

PerceptronDemo 0.08 0

Pierwszy parametr to krok korekcji, drugi wskazuje, czy użyć (*1*) czy nie użyć (*0*) biasu w uczonym perceptronie.

Znaczenie biasu

Bias spełnia w perceptronie ważną rolę. Rozważmy to na przykładzie. Na płaszczyźnie wzór na powierzchnię decyzyjną perceptronu można przedstawić jako:

$$w1*x+w2*y+w0=0$$

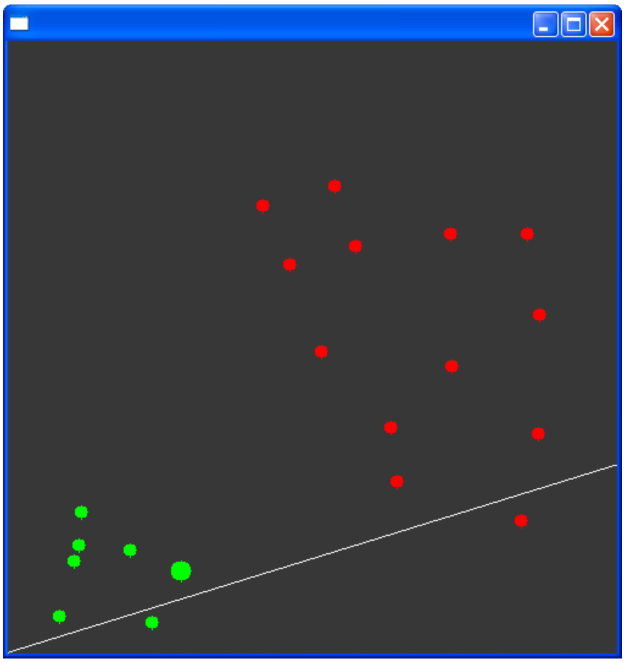
gdzie ***w0*** jest biasem. Jeśli wyznaczymy zależność ***y*** względem ***x****,* otrzymamy:

$$y=-(w1/w2)*x-(w0/w2)$$

Jest to znany wzór na prostą typu:

$$y=a*x+b$$

Wiadomo, że ***a*** jest współczynnikiem kierunkowym prostej (tangens kąta między tą prostą a osią *x*), natomiast ***b*** wyznacza punkt przecięcia przez prostą osi ***y***. Zwróćmy uwagę, że jeśli bias jest równy zero, to ***b*** również bedzie zero. Jeśli więc nasz perceptron nie będzie używał biasu, będzie on mógł wyznaczać granice decyzyjne w postaci prostych przechodzących przez początek układu współrzędnych (w przypadku większej liczby wymiarów sytuacja jest analogiczna). Nie będzie wiec mógł wyznaczać dowolnego podziału przestrzeni ani nauczyć sie dobrze klasyfikować danych z dwóch klas, nawet jeśli te klasy są liniowo separowalne (tzn. istnieje taka prosta – hiperpłaszczyzna – która je oddziela). Na rysunku poniżej widać przykład dwóch klas, których perceptron bez biasu nie jest w stanie się nauczyć – można to poznać po tym, ze każda linia reprezentująca wagi perceptronu, przechodzi przez punkt (0,0), czyli lewy dolny róg okna programu.



Rysunek 8: Przykład danych liniowo separowalnych, ktorych nie może nauczyć się perceptron bez biasu.

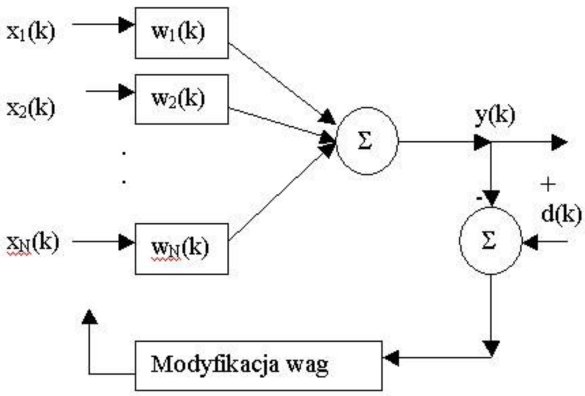
Pytania:

1. Czy w przypadku danych uczących, które nie są linowo separowalne, perceptron jest w stanie się odpowiednio nauczyć klasyfikować dane? Dlaczego? Przygotuj i przedstaw takie dane.
2. Przyglądnij się procesowi nauki perceptronu dla różnych wartości parametru ***eta***. Jak zależy proces uczenia od wartości tego parametru?
3. Uruchom program bez wykorzystania biasu i pokaż na przykładzie danych, które są liniowo separowalne, że perceptron bez biasu mimo tego nie daje rady się ich nauczyć.

4. Zapisz regułę uczenia perceptronu dla zastosowanej w programie bipolarnej funkcji aktywacji perceptronu.

Układy typu ADALINE

Układ typu **ADALINE (ang.** ***Adaptive Linear Neuron*****)** został zaproponowany w 1960 roku przez Widrowa i Hoffa. Nazywany jest również adaptacyjnym liniowym sumatorem ważonym. Model ważonego sumatora linowego jest przedstawiony na rysunku 9.



-

Rysunek 9: Schemat modelu ADALINE.

Algorytm modyfikacji wag ma charakter uczenia pod nadzorem. Sygnał wyjściowy ***y*** sumatora porównywany jest z sygnałem wzorcowym ***d***. Uzyskany w ten sposób błąd sumatora wykorzystany jest do zmodyfikowania wartości wag zgodnie ze wzorem:

$$w(k+1) = w(k) + \eta\, x(k)[d(k) - w^T(k)x(k)]$$

Parametr eta jest jak poprzednio krokiem korekcji. Algorytm ten realizuje uczenie pod nadzorem i należy do klasy algorytmów gradientowych, gdyż minimalizuje funkcję błędu neuronu wyrażoną wzorem:

$$E = (d(k) \ - y \ (k))^2$$

dla k-tego wzorca uczącego. W zależności od przyjętej metody uczenia, adaptacji wag dokonywać można po prezentowaniu każdej pary uczącej, lub też po prezentacji wszystkich par danych uczących. W drugim przypadku minimalizowaną funkcję błędu można zapisać jako:

$$E = \sum_{k=1}^{p} (y(k) - d(k))^2$$

Obliczając gradient funkcji błędu względem wag otrzymuje się:

$$\frac{\partial E}{\partial w(k)} = -2[d(k) - w^T(k)x(k)]x(k)$$

Gradient wyznacza kierunek największego wzrostu wartości funkcji, modyfikacji wag dokonuje się zatem w

przeciwnym kierunku, czyli kierunku największego spadku.

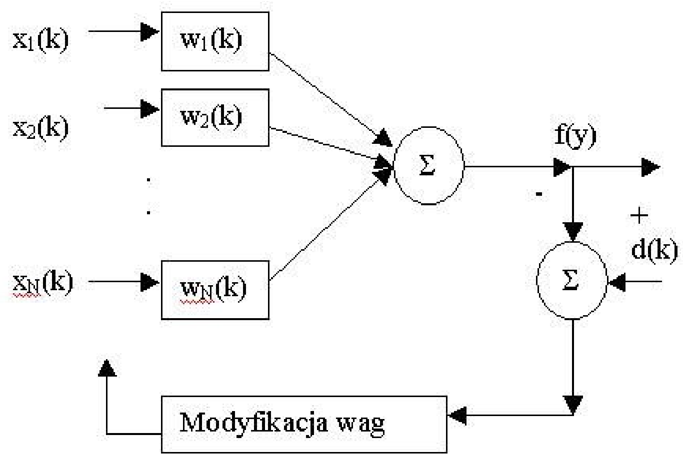
Adaptacyjny liniowy sumator ważony z nieliniową funkcją aktywacji.

W przypadku zastosowania nieliniowej funkcji aktywacji zgodnie z rysunkiem 10, konieczne jest zmodyfikowanie wzoru modyfikacji wag. Wynika to z innej postaci gradientu funkcji błędu. Gradient funkcji błędu wynosi:

$$\frac{\partial E}{\partial w(k)} = -2f'(y(k))x(k)[d(k) - y(k)]$$

Pojawia się pochodna funkcji aktywacji. Wzór na modyfikację wartości wag przyjmuje postać:

$$w(k+1) = w(k) + \eta x(k)[d(k) - y(k)]f'(y(k))$$



Rysunek 10 Schemat sumatora ważonego z nieliniową funkcją aktyacji.

Wartość pochodnej dla danej nieliniowej funkcji aktywacji, często można wyliczyć na podstawie wartości samej funkcji. Przykładowo, dla funkcji sigmoidalnej z parametrem ***beta*** równym 1, wartość pochodnej określa wzór:

$$f'(x) = f(x)(1 - f(x))$$

W przypadku wielu sumatorów (neuronów) połączonych w jedną warstwę, wagi każdego neuronu modyfikuje się osobno zgodnie z podanymi wzorami.

Algorytm wstecznej propagacji błędów

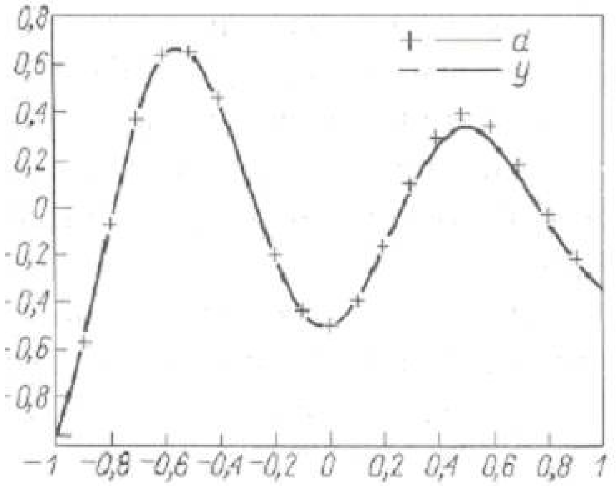
W przypadku sieci wielowarstwowych najczęściej stosowanym algorytmem uczenia jest algorytm wstecznej propagacji błędów. Jego nazwa pochodzi stąd, iż po obliczeniu odpowiedzi sieci na zadany wzorzec, obliczana jest wartość gradientu funkcji błędu dla neuronów ostatniej warstwy. Następnie modyfikuje się ich wagi. Błąd jest propagowany do warstwy wcześniejszej (przedostatniej). Wartości funkcji gradientu dla neuronów z tej warstwy obliczane są w oparciu o gradienty dla neuronów z warstwy następnej (czyli ostatniej). Modyfikowane są wagi kolejnej warstwy. Postępowanie trwa aż do warstwy wejściowej.

Istnieje wiele wariantów i ulepszeń oryginalnej metody wstecznej propagacji błędów. Pozwalają one uzyskać lepsze efekty uczenia lub przyśpieszyć proces nauki.

Zależność efektów uczenia sieci od jej architektury

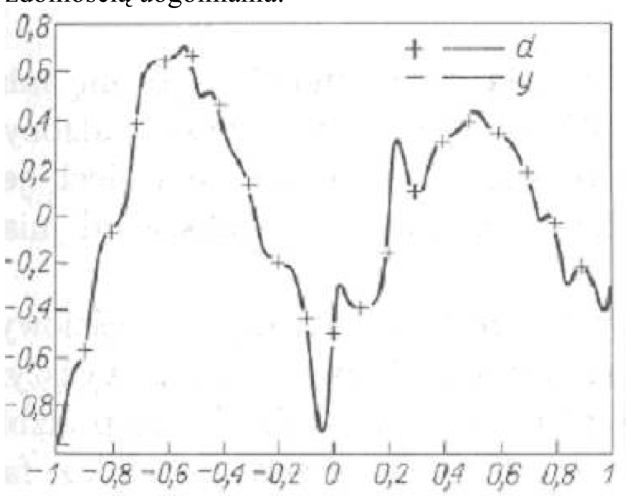
Udowodniono, że perceptron wielowarstwowy jest doskonałym aproksymatorem, tzn. może nauczyć się dowolnego odwzorowania (więc również aproksymować dowolną funkcję) pod warunkiem, że posiada odpowiednią liczbę neuronów.

Dobór odpowiedniej struktury sieci neuronowej, dostosowanej do danego problemu, jest zadaniem trudnym. Zły dobór parametrów sieci powoduje, że sieć nie może się dobrze nauczyć odpowiedniego odwzorowania, lub też charakteryzuje się złą zdolnością generalizacji, czyli źle radzi sobie z danymi, których wcześniej nie widziała (tzn. których nie użyto w trakcie uczenia sieci). Przykładowo, na rysunku 11 widzimy sieć dobrze aproksymującą zadaną funkcję:



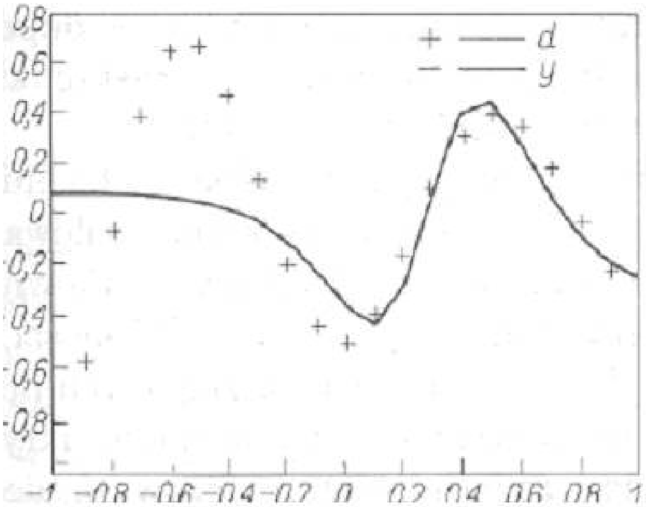
Rysunek 11. Dobrze odwzorowana funkcja.

Danymi uczącymi sieć były punkty oznaczone krzyżykami. Widać, że sieć odpowiada również dobrze w punktach, które nie zostały jej zaprezentowane w czasie nauki. Jednak nie zawsze nam się udaje tak dobrze nauczyć siec. Na rysunku 12 widzimy przypadek, gdy sieć została przeuczona i charakteryzuje się złą zdolnością uogólniania:



Rysunek 12. Przeuczona sieć.

Jednak jeśli na przykład użyjemy zbyt małej liczby neuronów, sieć będzie niedouczona (nie poradzi sobie z zadanym problemem, będzie miała zbyt małą moc obliczeniową) – rysunek 13.



Rysunek 13 Niedouczona sieć.

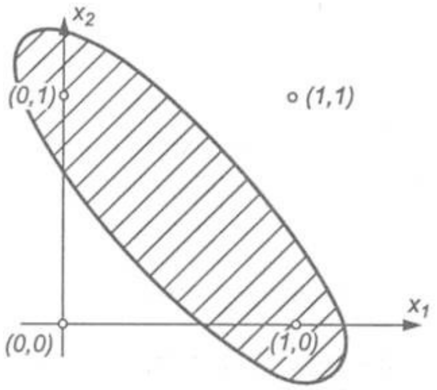
Podczas uczenia sieci dla danego zadania należy określić:
- architekturę sieci
- algorytm uczenia i jego parametry,
- dobór danych uczących i ich obróbkę wstępną
- liczbę prezentacji (iteracji) danych uczących (moment zakończenia nauki)
- metodę oceny jakości sieci
- początkowe wartości wag itd.

Symulacja układu logicznego XOR

Operacja logiczna XOR przyjmuje dwa parametry binarne i odpowiada również wartością 0 lub 1. Poniższa tabelka przedstawia odpowiedzi funkcji XOR dla wszystkich możliwych danych wejściowych:

X	**Y**	**XOR**
1	1	0
1	0	1
0	1	1
0	0	0

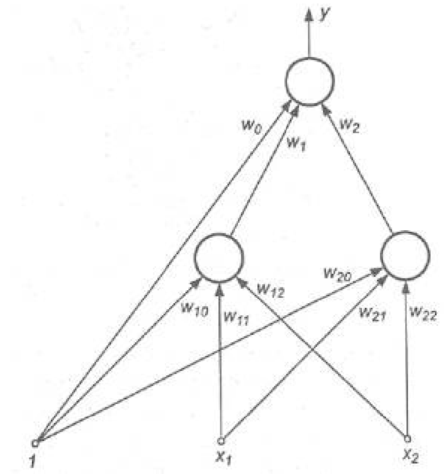
Problem polega na odwzorowaniu tej funkcji za pomocą sieci neuronowej (rysunek 14).



Rysunek 14. Problem XOR.

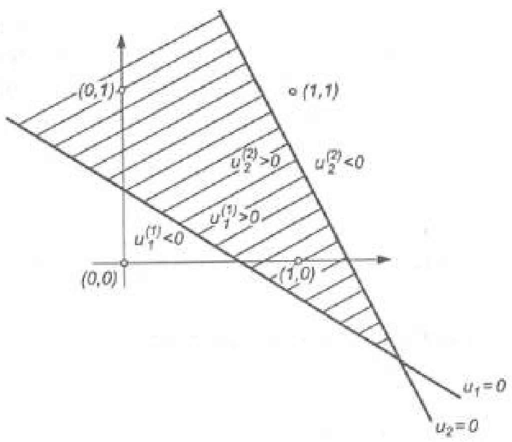
Wejściami dla sieci są wartości logiczne ***X*** oraz ***Y***, natomiast na wyjściu pojawia się wartość funkcji **XOR** dla podanych argumentów. Rozpatrywana sieć ma dwa sygnały wejściowe i jeden wyjściowy. Nie można jednak rozwiązać tego problemu za pomocą jednego neuronu o dwóch wejściach (plus bias), gdyż może on

wygenerować jedynie hiperpłaszczyznę, tj. linię prostą, dowolnie przysuniętą. Jedną linią nie można oddzielić odpowiednio od siebie argumentów funkcji XOR (czyli tych, które dają w wyniku 1 od tych które dają w wyniku 0). Dla rozwiązania tego klasycznego problemu potrzebna jest sieć z co najmniej dwiema warstwami. Model taki pokazano na rysunku 15.



Rysunek 15. Sieć rozwiązująca problem XOR.

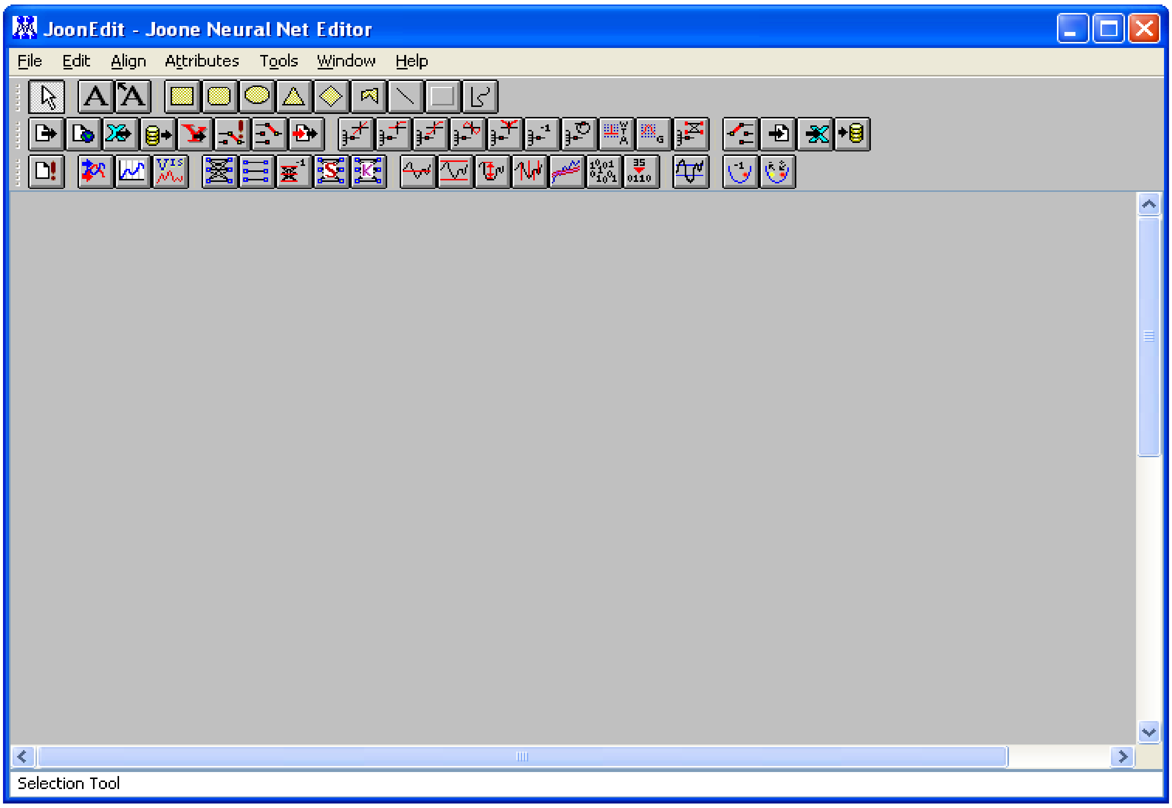
Rozpatrywana sieć dokonuje odpowiedniego nieliniowego podziału wykorzystując dwie hiperpłaszczyzny, jak pokazano na rysunku 16:



Rysunek 16. Rozwiązanie zadania XOR.

Środowisko JOONE

Do dalszej pracy wykorzystamy środowisko ***JOONE (Java Object Oriented Neural Engine)***. Jest ono dostępne pod adresem http://www.joone.org/. Daje ono szerokie możliwości konstruowania i testowania różnorodnych architektur sieci neuronowych w zastosowaniu do szeregu problemów. Na rysunku 17 przedstawiono interfejs środowiska.

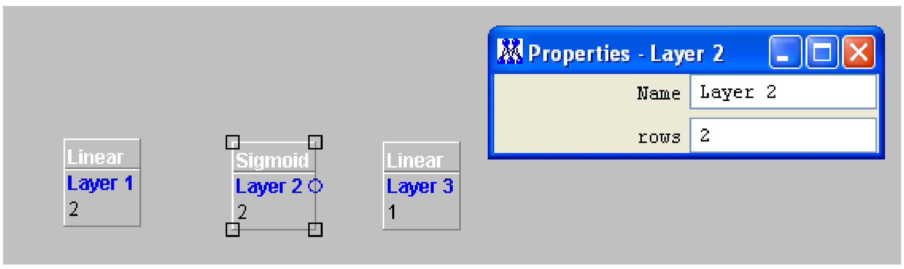


Rysunek 17: JOONE (Java Object Oriented Neural Engine).

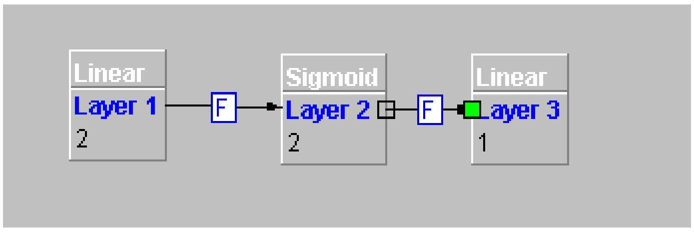
W celu rozwiązania problemu XOR oraz zapoznania się z Joone przed przystąpieniem do bardziej skomplikowanych zadań, skonstruujemy sieć neuronową typu wielowarstwowego perceptronu, która powinna rozwiązać ten problem.

Tak jak zostało pokazane wcześniej, nasza sieć musi mieć warstwę ukrytą i wyjściową. W Joone będzie jeszcze dodatkowa warstwa wejściowa, której zadaniem będzie jedynie doprowadzanie sygnałów wejściowych do warstwy ukrytej.

Podstawowymi elementami, z których budujemy sieć w Joone jest warstwa (*ang. Layer*). Przyjrzyj się panelowi Joone i znajdź na toolbarze odpowiednie przyciski: ***NewLinearLayer, NewSigmoidLayer.*** Nastepnie dodaj trzy warstwy tak jak poniżej:



Pierwsza warstwa powinna mieć dwa neurony (parametr ***rows;*** w warstwie wejściowej jest tyle neuronow, ile jest wymiarów danych wejściowych), druga warswa (warstwa ukryta) jest warstwą neuronów z sigmoidalnymi funkcjami aktywacji i posiada dwa neurony, warstwa wyjściowa ma jeden liniowy neuron. Przeciągając linię między warstwami, połącz warstwy w taki sposób, aby otrzymać następującą strukturę:



Każdy neuron z danej warstwy został połączony z każdym neuronem warstwy następnej. Struktura sieci jest gotowa. Teraz przygotujemy dane uczące. Stwórz plik tekstowy ***xor.data*** z następującą zawartością:

0.0;0.0;0.0

0.0;1.0;1.0

1.0;0.0;1.0

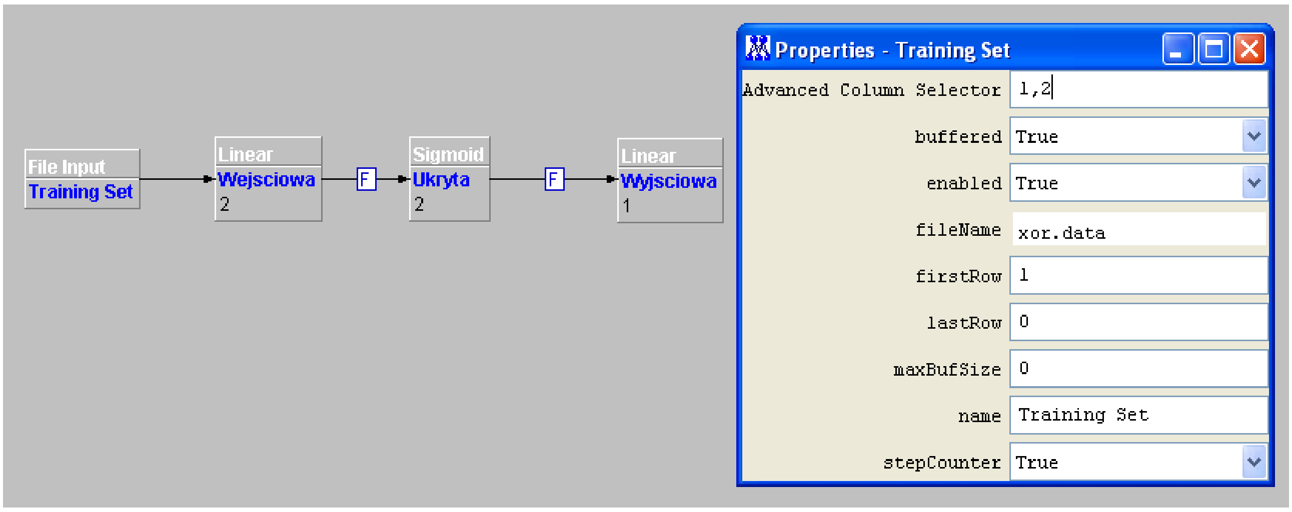
1.0;1.0;0.0

Dwie pierwsze kolumny reprezentują dane wejściowe, trzecia kolumna to wzorcowe odpowiedzi. Dodaj w edytorze strumień wejściowy z pliku: ***NewFileInputLayer*** . Kliknij na niego prawym przyciskiem myszy i ustaw jego właściwości (p. rysunek poniżej):

Advanced Column Selector: 1,2

fileName: xor.data

Co oznacza, ze dane wejściowe to kolumny 1 oraz 2.



Połącz dane wejściowe z warstwą wejściowa.

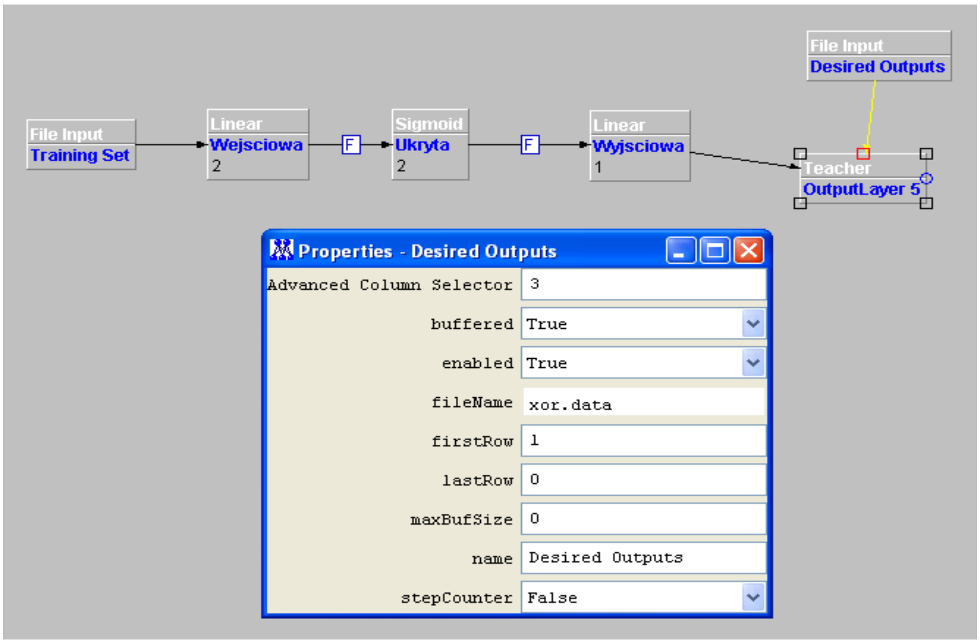
Dodajemy nauczyciela, który bedzie czuwał nad procesem uczenia. Dodaj ***NewTeacherLayer*** oraz jeszcze jeden ***NewFileInputLayer***. Ustaw właściwości nowej warstwy ***FileInput*** na:

Advanced Column Selector: 3

fileName: xor.data

Znaczy to, że wzorcowymi danymi wyjściwoymi sieci są dane z trzeciej kolumny pliku ***xor.data***.

Połącz nowe elementy jak na rysunku poniżej.



Sieć jest gotowa do nauki. Otwórz ***Tools->ControlPanel*** i ustaw wartości na:

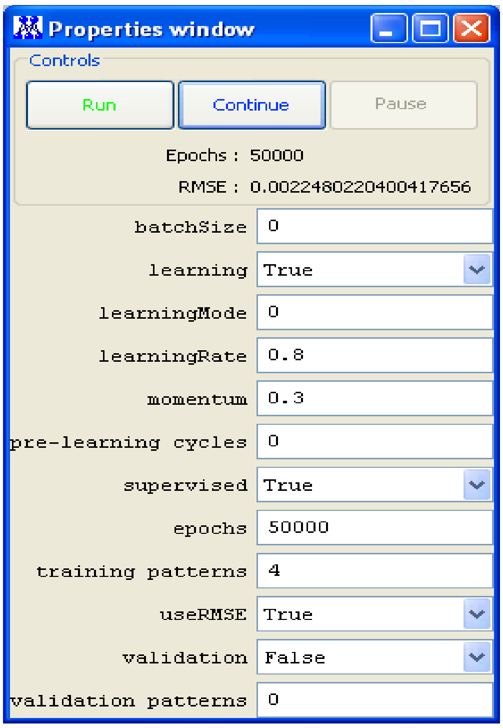
learning: True

learningRate: 0.8

momentum: 0.3

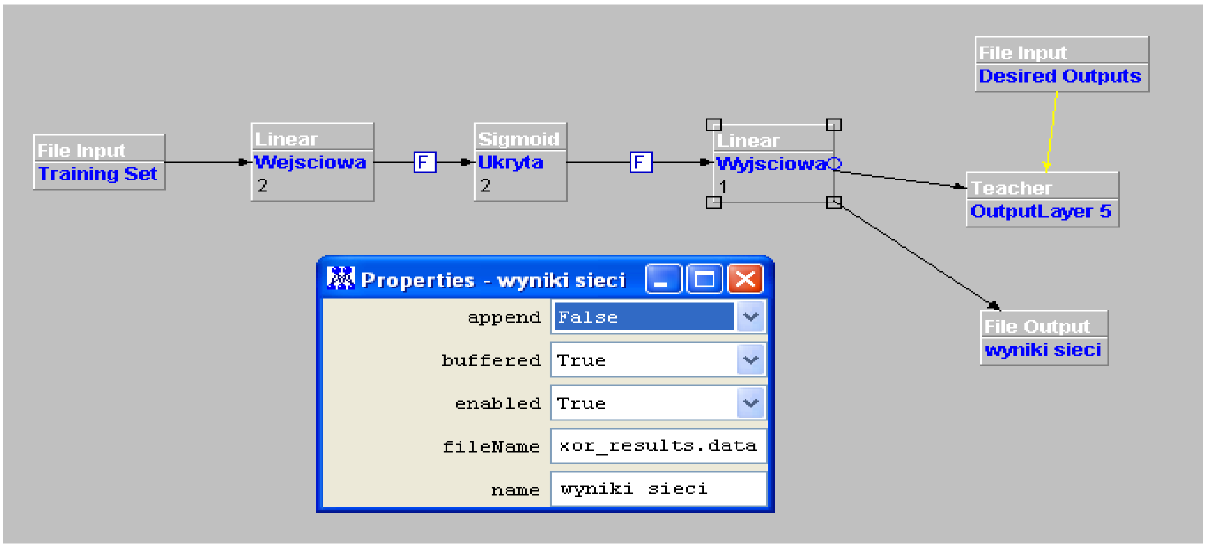
epochs: 50000

training patterns: 4



Kliknij ***Run.*** Obserwuj, jak błąd średniokwadratowy (RMSE) maleje.

Czas przetestować sieć. Dodaj do edytora ***NewFileOutputLayer*** i ustaw jego włąsciwości jak na rysunku.



Ustaw właściwość ***Teacher*** **enabled** na **false.** Następnie przejdź do **ControlPanel** i ustaw:

learning: False

epochs: 1

training patterns: 4

Zawartość pliku ***xor_results.data*** powinna wyglądać mniej wiecej następująco:

0.005218926410945011

0.9999885110810375

0.9999902774208039

0.003631796924953539

Widać, że sieć nauczyła się odpowiadać odpowiednimi wartościami (bliskimi 1 lub bliskimi 0) na odpowiednie dane wejściowe. Porównaj otrzymane odpowiedzi z odpowiedziami wzorcowymi dla problemu XOR.

Nauczonż sieć można zapisać wybierając ***File->Save***.

<u>Zadania</u>:

1. Sprawdź, że rzeczywiście sieć posiadająca jeden neuron w warstwie ukrytej nie jest w stanie nauczyć sie problemu xor. Po czym można to poznać?
2. Przetestuj proces uczenia sieci dla różnych ustawień parametrów uczenia: ***learningRate, momentum, liczba iteracji (epochs).*** Do ilu iteracji warto uczyć sieć (do kiedy błąd spada)?
3. Co się stanie jeśli zwiekszymy libczę neuronów ukrytych na 3, 4, 5, 6 itd. Czy przyśpieszy to proces uczenia lub zmniejszy błąd? Zawsze?

4. Powtórz poprzednie zadania z wykorzystaniem warstwy ukrytej z neuronami typu liniowego. Czy istnieje zauważalna rożnica w porównaniu do zastosowanych wczesniej neuronów typu sigmoidalnego?
5. Dla jakiej architektury sieci błąd był najmniejszy? Po jakiej liczbie iteracji i z jakimi parametrami? Przedstaw dane w tablekach.
6. Czy problemy logiczne AND oraz OR są problemami liniowo separowalnymi? Przygotuj odpowiednie dane uczące i sprawdz swoje przypuszczenia powtarzając eksperymenty dla tych nowych danych. Odpowiedź usasadnij posiłkując się wynikami.

Wszystkie wnioski i spostrzeżenia poprzyj w sprawozdaniu wynikami przeprowadzonych obliczeń i symulacji, jak również przedstaw przeprowadzane eksperymenty i badane sieci na rysunkach („screenach”). Wyniki przedstaw w opisanych tabelkach.